

Communication Aspects of an Asynchronous Parallel Evolutionary Algorithm

Pu Liu and Michael J. Lewis

Department of Computer Science
State University of New York (SUNY) - Binghamton
Binghamton, NY 13902 USA

Abstract. *Parallel evolutionary algorithms imitate evolution in nature in order to solve complex problems, including function optimization. Most such algorithms do not do a good job of overlapping communication and computation, requiring either barrier synchronization among processes, or a centralized process that manages information exchange. This limits the size of problems that can be solved, and compromises the efficiency of the algorithms. We have designed a fully distributed asynchronous parallel evolutionary algorithm (APEA) [13], which has proven extremely effective. For one well-known function optimization problem (G10 [16, 19]) we found a new best solution, and for another (BUMP [8]) we solved the 1,000,000 dimension problem, when only the 50 dimension problem had previously been solved. This paper describes the communication characteristics of our algorithm, focusing on two components: (1) the elimination of synchronization and centralization, and (2) the setting of parameters to control the amount of communication generated by the algorithm, which influences both its efficiency and its effectiveness in finding optimal solutions.*

1. Introduction¹

Evolutionary algorithms imitate nature's complex processes to solve difficult problems, including function optimization problems. In general, a function optimization problem attempts to find the values for a vector $X = [x_1, x_2, \dots, x_n]$ that optimize $F(X)$ according to some criteria (e.g. minimum or maximum), subject to certain constraints on the vector components of X . The approach is to generate a set of vector solutions, or *individuals*, distributed across multiple processes, called subpopulations or *demes*. Each deme uses its individual solutions to create a new solution, which it adds to the subpopulation space, replacing some other individual. This new *offspring* individual can be generated using one of

several different strategies, including for example, a multiparent crossover method [5], which randomly weights multiple individuals and combines them to produce a new one. The demes continue to generate new offspring individuals in this way, replacing the most "unfit" individual during each iteration. The algorithm terminates when either a "satisfactory" solution is found, a fixed time limit is reached, or a subpopulation evolves to the point where all of its individuals are the same.

To avoid converging too soon (i.e. before an optimal solution is found), demes periodically exchange individuals with one another. This allows more individuals to affect the convergence and influence the final solution that is reached. In a parallel algorithm, this individual exchange is the source of communication. Thus, a deme can replace an individual by either generating a new one based on other individuals it contains, or by receiving one from a neighboring deme. The frequency with which individuals are exchanged, compared to how often they are generated locally, determines the granularity of the algorithm. Tradi-

1. This work was partially supported by National Science Foundation (NSF) Instrumentation Grant EIA9911099.

tionally, parallel evolutionary algorithms are synchronous. Iterations proceed in lock-step with one another, as the algorithm alternates between communication and computation phases. Those algorithms that are asynchronous use a centralized repository of individuals to facilitate exchange. As we show in Section 3, barrier synchronization and centralization are unnecessary. Our fully distributed asynchronous algorithm allows processes to proceed at their own pace, without waiting for others to complete an individual exchange phase. The scalability, efficiency, and fault tolerance of the algorithm are directly related to making communication completely asynchronous.

The frequency of exchange also affects the quality of the solution that the algorithm generates. Unlike many other algorithms, it is not always desirable to converge as quickly as possible, because this may cause a suboptimal solution to be found. In particular, this can happen when individuals are exchanged too infrequently. But when individuals are exchanged too frequently, the communication begins to dominate and the algorithm becomes less efficient, as it takes longer to generate new offspring individuals (because they have to arrive from other demes, rather than being produced locally). As we will show, this is a much bigger problem for synchronous algorithms. In general, it is not known how to determine the optimal frequency of individual exchange that causes the algorithm to converge at the best possible solutions. Therefore, experimental results are required to help provide guidelines for programmers and users of the algorithm.

We have introduced our fully distributed asynchronous parallel evolutionary algorithm (APEA) in [13], presenting results that demonstrate the efficiency and effectiveness of the algorithm. In particular, we report a new optimal solution for G10, and present a solution to the 1,000,000 dimension BUMP problem, when only the 50 dimension problem had been solved in the past.

In this paper, we discuss the communication aspects of our APEA, focusing on (1) the elimination of synchronization and centralization in the algorithm, and (2) some initial results that begin to indicate how best to set the parameters of the

algorithm to determine the rate of communication that produces the best results. These aspects are mentioned only briefly in [13]. Section 2 describes traditional synchronous algorithms and centralized asynchronous algorithms. Then Section 3 presents our APEA, which uses only non-blocking communication primitives of MPI [17] and PVM [18] (we have implemented our algorithm using both message passing systems), and allows demes to exchange individuals directly. We then recount several results that begin to help determine the best rate of individual exchange for a few sample function optimization problems. We summarize and discuss directions for future work in Section 5.

2. Synchronous and Centralized Algorithms

As described above, traditional parallel evolutionary algorithms, which use the *island model* for organization, are synchronous. Demes coordinate to exchange individuals in lock-step with one another. Each deme completes its first *epoch* of computation, selects a set of other subpopulations to which some number of individuals are sent, and receives individuals from some set of other subpopulations. Once each deme has completed the exchange, it can proceed to the next epoch. Clearly, this approach allows the slowest processor during each epoch to determine the overall progress of the algorithm.

An alternative *master-slave* approach eliminates the requirement that demes synchronize and proceed in lock-step with one another, but introduces a centralized repository through which all individual migration occurs. In this approach, a master process is responsible for generating the initial population and performing reproduction. Slave processes receive subpopulations from the master, perform crossover and migration, and then evaluate the fitness of the individuals in the resulting population. The “best” individuals are returned to the master, along with their fitness values, and a new subpopulation is returned for evaluation. Clearly, this centralized control of the algorithm limits its scalability.

3. A Fully Distributed APEA

For many parallel computations, a synchronous approach is necessary. This is true when processes require data *from a specific iteration* from other neighboring nodes. But this constraint is not present in parallel evolutionary algorithms for function optimization. Demes don't care when (i.e. during which epoch) individuals from other subpopulations were generated. Therefore, it is perfectly acceptable for some demes to proceed through iterations more quickly than others. The only danger is that if communication does not take place frequently enough with some deme, then that deme will converge to a suboptimal solution too quickly. This problem does not exist in synchronous algorithms because demes wait for one another before proceeding.

Below, we show the algorithm each process uses:

```
PROCESS G
1. t=0
2. Initialize P(t)={X1, X2, ..., Xn}
3. Evaluate P(t)
4. While (not terminated) do
5.   If (any message arrived) then
6.     Xnew = Recv_Individual()
7.   else Xnew = Local_Generate()
8.   P'(t) = P(t) ∪ {Xnew}
9.   P(t) = select N best individuals from P'(t);
10.  Locate Xbest in P(t);
11.  If (t mod T = 0) and (Xbest changed) then
12.    Send Xbest to Q other process(es)
13.  t = t + 1
14. End While
```

The difference between this algorithm and synchronous algorithms is in the implementation of communication operations. Lines 5 and 6 employ the `MPI_IProbe()` (or `pvm_probe()`) message to check whether individuals have arrived from other populations. If so, they can be received and incorporated into the local subpopulation. But if not, the local deme can continue to generate individuals locally (in Line 7) without stopping to wait for other processes. Likewise, the `send()` operation in Line 12 is non-blocking. The algorithm does not require centralized control of individual exchange.

3.1 Discussion

As reported in [13], with this algorithm we have found an already-known best solution to well-known function optimization problems G7 [16, 19] and H1 [5], found a new best solution for G10 [16, 19], and solved the scalable BUMP problem for dimension 1,000,000, when the largest problem previously reported solved had dimension 50.

The asynchronous nature of the algorithm improves its fault tolerance. Because the demes are looking for a single best solution, and none of the communication operations block to wait for other demes, one or more of the processes representing the subpopulations can be lost without affecting the progress of the rest of the processes. In a synchronous version of the algorithm, the loss of a single deme can cause others to block indefinitely on a synchronous communication call. This feature of the algorithm clearly improves its scalability, as an increase in the number of processes necessarily results in an increase in the likelihood that one or more of them will fail.

Another benefit that asynchronous communication affords is the ability to be effective in environments where the load is not well balanced. Rather than allowing the slowest processor to determine the progress of the algorithm, faster processes can continue to make progress without input from other demes. Eventually, without enough individual exchange, they may converge on solutions too quickly; the details of this effect are not yet well understood.

4. Controlling the Amount of Communication to Achieve Better Solutions

As alluded to earlier, parallel evolutionary algorithms, including our distributed asynchronous algorithm, can be controlled by two different parameters. Parameter Q indicates the number of other processes with which one process communicates when a new best individual is found within its deme. Parameter T determines the frequency of individual migration. Together, Q and T determine the total communication cost of each iteration of APEA; the larger the value of Q, and the

smaller the value of T , the more communication per computation in the algorithm. The rate of migration of individuals, as compared to the rate of generating new individuals locally within a deme, affect the solution at which the algorithm converges. Unfortunately, it is not known precisely what the effect of these parameters is. We have performed some preliminary tests that indicates that for some problems increasing communication increases the duration of the algorithm and allows it to converge on better solutions. For other problems and results, the amount of communication does not seem to directly correlate with the quality of solution that is achieved.

Table 1 below shows the results of G7, with

TABLE 1. Results of G7 with various amounts of communication vs. computation

NP	T	Q	Med. Time	Solution
1	0	0	39	24.30620906905601
2	1	2	18	24.30620906854010
6	30	6	24	24.30620906841773
4	30	4	21	24.30620906839757
12	30	12	33	24.30620906837152
8	30	8	28	24.30620906834294
12	1	12	35	24.30620906834074
8	1	8	26	24.30620906831426
16	1	16	43	24.30620906830320
10	1	10	32	24.30620906828532

various amounts of communication vs. computation. The rows are organized according to increased quality of solution (G7 is a minimization problem). We notice that the sequential version converges at the worst solution, followed by the version that communicates individuals with only two other demes. Among the other eight runs, the four that communicated most frequently ($T=0$) performed better than any of the four that

communicated every 30 iterations ($T=30$). The best solution was achieved with 10 processors communicating with one another, rather than with 12 or 16, however, showing that more communication (and processors) do not necessarily lead to a better solution.

We also ran experiments on the 1000 dimension BUMP problem, fixing Q and varying T , and then fixing T and varying Q . The results are shown in Tables 2 and below, again sorted by quality of solution.

TABLE 2. BUMP problem, $T=200$, $n=1000$

Q	Time	Solution
2	14653	0.84421
2	14115	0.84421
2	93010	0.84265
1	93010	0.84265
8	28877	0.84259
8	27794	0.84218
6	3996	0.84218
6	4246	0.84218
8	30169	0.84193
1	28016	0.84163
4	5187	0.83924
4	4998	0.83889
1	21196	0.83889
6	3130	0.83376

It is difficult to draw any conclusions about the effect of communication based on Table 2. The best solution is achieved when $Q=6$, but the other two runs with this value of Q converge on a worse solution. Setting $Q=8$ (the most communication) resulted in solutions that were in the middle of the pack, as compared to the others, and setting $Q=1$ (the least communication) resulted in some good solutions and some worse solutions.

In Table 3, we notice that three of the best

TABLE 3. BUMP problem, n=1000, Q=2

T	Iterations	Solution
300	12389	0.8448539
300	12095	0.8448454
500	13081	0.8442194
300	10869	0.8442193
500	14330	0.8442193
200	14653	0.8442193
200	14115	0.8442193
200	9301	0.8426541
500	13105	0.8421879
400	11720	0.8421879
400	11329	0.8410598
100	10969	0.8410598
100	11830	0.8410597
100	8867	0.8410594
400	10899	0.8410299

four solution are achieved when the most communication is induced (T=100). The next best performing value of T is 400, and it is difficult to infer any further generalizations from the results.

The results we have attained thus far should be considered nothing more than anecdotal evidence that increased communication may lead to better quality solutions, generally speaking. A much more comprehensive study, which we plan to undertake, is required to determine the effect of the amount of communication on the convergence and quality of solutions. This study should include other function optimization problems, and many more runs of the algorithm.

5. Summary and Future Work

We have described a fully distributed asynchronous parallel evolutionary algorithm that we have applied to function optimization problems. We focus on the communication aspects of the algorithm, and describe how our asynchronous version is more fault tolerant and scalable than synchronous or centralized versions. We describe how two parameters, Q and T, can be altered to affect the amount of communication that the algo-

rithm introduces, and report a small set of preliminary results that seem to suggest indicate that increasing communication increases the quality of solutions, to a certain extent.

We plan to study our distributed APEA in grid computing [11] environments that are characterized by heterogeneity in computing resources and communication link performance. We hope to continue to learn more about the effects of communication parameters T and Q on the convergence characteristics of the algorithm, the quality of solution that is achieved, and the algorithm's run time. The algorithm's scalability and resistance to faults will hopefully allow us to solve larger problems, and to find better solutions.

References

- [1] L. A. Anbarasu et al., "Multiple Sequence Alignment by Parallel Evolvable Genetic Algorithms," in A. S. Wh (ed.) Proceedings of the 1999 Genetic and Evolutionary Computation Conference Workshop Program, Orlando, Fla, July 13, 1999, pp. 154-156.
- [2] E. Cantu-Paz, "Markov Chain Models of Parallel Genetic Algorithms," IEEE Transactions on Evolutionary Computation, Vol. 4, No. 3, pp. 216-226, 2000.
- [3] M. A. El-Beltagy, et al. "Metamodeling Techniques For Evolutionary Optimization of Computationally Expensive Problems: Promises and Limitations." Genetic Algorithms and Classifier Systems, 1999.
- [4] P. B. Grosso, "Computer Simulations of Genetic Adaptation: Parallel Subcomponent Interaction in a Multi-locus Model," Ph.D. Dissertation, University of Michigan, 1985.
- [5] T. Guo, L. S. Kang, "A New Evolutionary Algorithm for Function Optimization," Wuhan University Journal of Natural Sciences, Vol. 4, No. 4, pp. 409-414, 1999.
- [6] A. J. Keane, "Experiences with Optimizers in Structural Design," in Proceedings of the Conference on Adaptive Computing in Engineering Design and Control 94, ed. (I. C. Parmee, Plymouth, 1994), pp. 14-27.
- [7] A. J. Keane, "A Brief Comparison of Some Evolutionary Optimization Methods," Proceedings of Applied Decision Technologies (Modern Heuristic Methods), 1995.
- [8] A. J. Keane, "Genetic Algorithm Optimization of Multi-peak Problems: Studies in Convergence and

Robustness.” *Artificial Intelligence in Engineering*, 1995.

[9] A. J. Keane, “Passive Vibration Control Via Unusual Geometric: Application of Genetic Algorithm Optimization to Structural Design,” *Journal of Sound and Vibration*, 1995.

[10] K. Kojima, W. Kawamata, et al, “Network Based Parallel Genetic Algorithm using Client-Server Model”, in *Proceedings of the Conference on Evolutionary Computation 2000*, pp. 244-250.

[11] I. Foster, C. Kesselman, (eds), *The Grid: Blueprint for a New Computing Infrastructure*, Morgan Kaufmann Publishers, 1998.

[12] H. Lienig, “A Parallel Genetic Algorithm for Performance-Driven VLSI Routing,” *IEEE Transactions on Evolutionary Computation*, Vol. 1, No.1, pp. 29-39, 1997.

[13] P. Liu, M. J. Lewis, F. Lau, C. Wang, “A New Asynchronous Parallel Evolutionary Algorithm for Function Optimization,” Submitted to PPSN 2002, March 2002.

[14] J. D. Lohn, “A Circuit Representation Technique for Automated Circuit Design,” *IEEE Transactions on Evolutionary Computation*, Vol. 3, No. 3, pp. 205-219, 1999.

[15] Z. Michalewicz and M. Schoenauer, “Evolutionary Algorithms for Constrained Parameter Optimization Problems,” *Evolutionary Computation*, Vol. 4, No. 1, pp. 1-32, 1996.

[16] Z. Michalewicz, S. Esguvel et al., “The Spirit of Evolutionary Algorithms.” *Journal of Computing and Information Technology*, Vol. 7, pp. 1-18, 1999.

[17] “MPI: A Message-Passing Interface Standard.” *Message Passing Interface Forum*, 1994.

[18] Sunderam, V. S., “PVM: A Framework for Parallel Distributed Computing.” *Concurrency: Practice and Experience*, Vol. 2, No. 4, pp. 315-339, December, 1990.

[19] M. J. Tahk, B. C. Sun, “Coevolutionary Augmented Lagrangian Methods for Constrained Optimization.” *IEEE Transactions on Evolutionary Computation*, Vol. 4, No. 2, 2000.

[20] A. Wu, K. Y. Wu, R. M. M. Chen, Y. Shen, “Parallel Optimal Statistical Design Method with Response surface modeling using genetic algorithms,” *Circuits, Devices and Systems, IEE Proceedings-*, Vol. 145, No.1, 1998.

[21] H. T. Yang, “A Parallel Genetic Algorithm Approach to Solving the Unit Commitment Problem: Implementation on the Transputer Networks,” *IEEE Transactions on Power Systems*, Vol. 12, No.2, pp. 661-668, 1997.